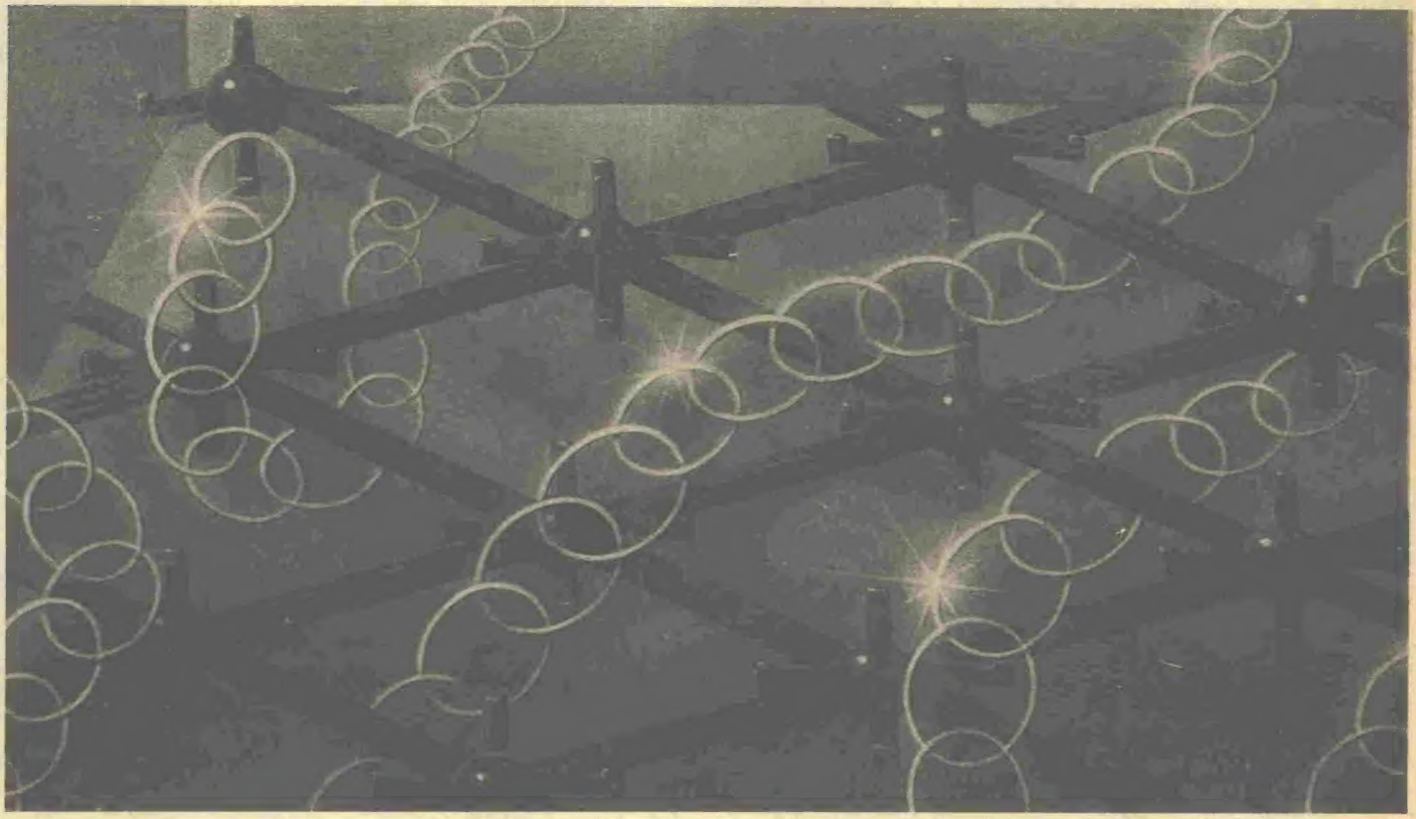
การจัดการ หน่วยความจำของดอส



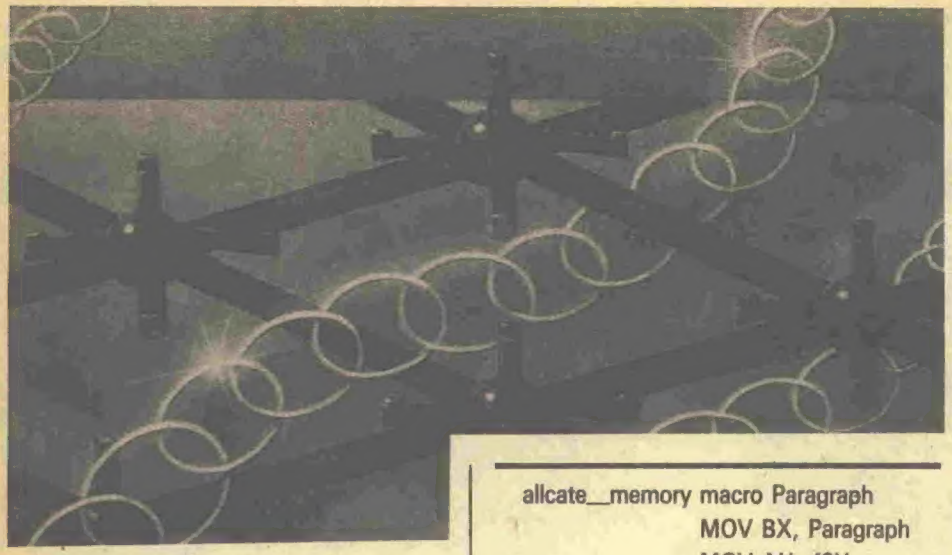
บุญภพ ลิมมโนกุล

บนเครื่องไมโครคอมพิวเตอร์ไอบีเอ็มที่มีหน่วยความจำ 640K ดอสจะทราบว่า ระบบมีการจัดสรรหน่วยความจำไว้อย่างไร โดยเฉพาะเมื่อมีการรันโปรแกรมพวกโปรแกรมเรสซิเดนต์ (Resident) ที่จะฝังอยู่ในหน่วยความจำ ดอสจะนำโปรแกรมเหล่านี้อ่านลงมาไว้ในแอดเดรสไหนของหน่วยความจำ และคงเหลือหน่วยความจำไว้ใช้งานอีกเท่าไร บทความนี้จะกล่าวถึงเรื่องที่เกี่ยวข้องกับฟังก์ชันของดอสที่เกี่ยวกับการจัดสรรหน่วยความจำ การเชื่อมโยงของบล็อคความจำ และการใช้โปรแกรมเทอร์โบเบสิกเข้าตรวจดูการจัดสรรหน่วยความจำ

ฟังก์ชันของดอสที่เกี่ยวกับการจัดสรรหน่วยความจำ

ดอสตั้งแต่เวอร์ชัน 2 เป็นต้นมาได้จัดการเกี่ยวกับพื้นที่หน่วยความจำที่จัดสรรให้ เรียกว่า เมมโมรีคอนโทรลบล็อก (Memory Control Block) หรือเขียนย่อ ๆ ว่า MCB ดอสจะเขียนไว้ที่ตำแหน่งเริ่มต้นของทุก ๆ ส่วนก่อนเริ่มต้นพื้นที่หน่วยความจำ รูปแบบหรือหน้าตาของ MCB นั้นจะขอยกไปในหัวข้อต่อไป

เรามาดูเกี่ยวกับฟังก์ชันที่ดอสได้กำหนดให้ใช้มีอยู่ 3 ฟังก์ชันคือ ฟังก์ชัน



48H, 49H และ 4AH แต่ละฟังก์ชันมีวิธีการใช้ฟังก์ชันเหล่านี้ผ่านทางอินเตอร์รัพต์ 21H ดังนี้

ฟังก์ชัน 48H

เป็นฟังก์ชันร้องขอให้ดอสจัดสรรหน่วยความจำเพิ่มให้กับโปรแกรม (Allocate Memory) โดยให้ค่ารีจิสเตอร์ AH=48H, BX= จำนวนพารากราฟ (1 พารากราฟ = 16 ไบต์) ที่ต้องการ ตามด้วยอินเตอร์รัพต์ 21H การตอบสนองจากดอสจะปรากฏให้เราตรวจสอบได้ที่แฟล็กตัวทด (Carry flag) ถ้าเป็นหนึ่ง (set) ให้ตรวจสอบดูที่รีจิสเตอร์ AX ค่ารีจิสเตอร์ AX มีค่าตัวเลขอยู่ 2 ตัวคือ 7 หรือ 8 ถ้าค่า AX เท่ากับ 7 แปลความว่า MCB นี้เสียหาย ปกติแล้วจะไม่ค่อยเกิดหากว่าเราไม่แตะต้องกับ MCB โดยตรง ถ้าค่า AX เท่ากับ 8 แปลความว่าหน่วยพารากราฟความจำที่เราขอใช้จากดอสโดยกำหนดให้กับรีจิสเตอร์ BX ดอสมีให้ไม่เพียงพอ แต่ดอสจะบอกว่าสามารถจัดสรรได้กี่พารากราฟ และตอบกลับมาทางรีจิสเตอร์ BX, แฟล็กตัวทดเป็นศูนย์ ค่า AX จะบอกเราว่า จุดเริ่มต้นอยู่ที่เซกเมนต์ใดในหน่วยความจำ แมคโครของฟังก์ชันนี้เขียนได้ดังนี้

```
allcate_memory macro Paragraph
               MOV BX, Paragraph
               MOV AH, 48H
               INT 21H
               endm
```

ฟังก์ชัน 49H

เป็นฟังก์ชันคืนการควบคุมหน่วยความจำให้กับดอส (Free Allocate memory) โดยให้ค่ารีจิสเตอร์ AH= 49H, ES = หมายเลขเซกเมนต์ที่เราต้องการปล่อยให้อิสระ ตามด้วยอินเตอร์รัพต์ 21H การตอบสนองจากดอสที่โปรแกรมเราตรวจสอบได้คือแฟล็กตัวทด ถ้าแฟล็กนี้เป็นหนึ่ง ให้ตรวจสอบดูที่รีจิสเตอร์ AX เท่ากับ 7 หมายถึง MCB เสียหาย ถ้า AX เท่ากับ 9 แปลความว่า เซกเมนต์ที่ต้องการให้ว่าง (Free) นั้นไม่ถูกต้อง ถ้าแฟล็กตัวทดเป็นศูนย์ไม่เกิดข้อผิดพลาด แมคโครของฟังก์ชันนี้เขียนได้ดังนี้

```
free_memory macro Segment_address
            MOV AX, Segment_address
            MOV ES, AX
            MOV AH, 49H
            INT 21H
            endm
```

ฟังก์ชัน 4AH

เป็นฟังก์ชันใช้ในการปรับขนาดบล็อกหน่วยความจำ (Set Block) ให้มีขนาดเล็กลงหรือเพิ่มขึ้น โดยให้ค่ารีจิสเตอร์ AH = 4AH, BX = ขนาดความจำที่ลดลง, เพิ่มขึ้นเป็นกี่พารากราฟ, ES = เซกเมนต์แอดเดรสบล็อกของหน่วยความจำตามอินเตอร์รัพต์ 21H การตอบสนองจากดอสที่โปรแกรมเราตรวจสอบได้คือ แฟล็กตัวทด ถ้าแฟล็กนี้เป็นหนึ่ง ให้ตรวจสอบต่อไปที่รีจิสเตอร์ AX ถ้า AX เป็น 7 หมายถึง MCB เสียหาย, AX = 8 หมายถึงความจำส่วนที่ยังว่างอยู่นั้นไม่เพียงพอ พร้อมกับดอสจะบอกว่าจำนวนความจำจะจัดให้ได้สูงสุดกี่พารากราฟ สุดท้ายค่า AX = 9 หมายถึงเซกเมนต์แอดเดรสที่เราส่งให้ดอสผ่านทางรีจิสเตอร์ ES ในตอนแรกไม่ถูกต้อง แต่ถ้าแฟล็กตัวทดเป็นศูนย์ แสดงว่ามีการทำงานสมบูรณ์ แมคโครของฟังก์ชันนี้เขียนได้ดังนี้

```
Set_block macro last_para, Segment_address
MOV AX, Segment_address
MOV ES, AX
MOV BX, last_para
MOV AH, 4AH
INT 21h
endm
```

การเชื่อมโยงของบล็อกความจำ

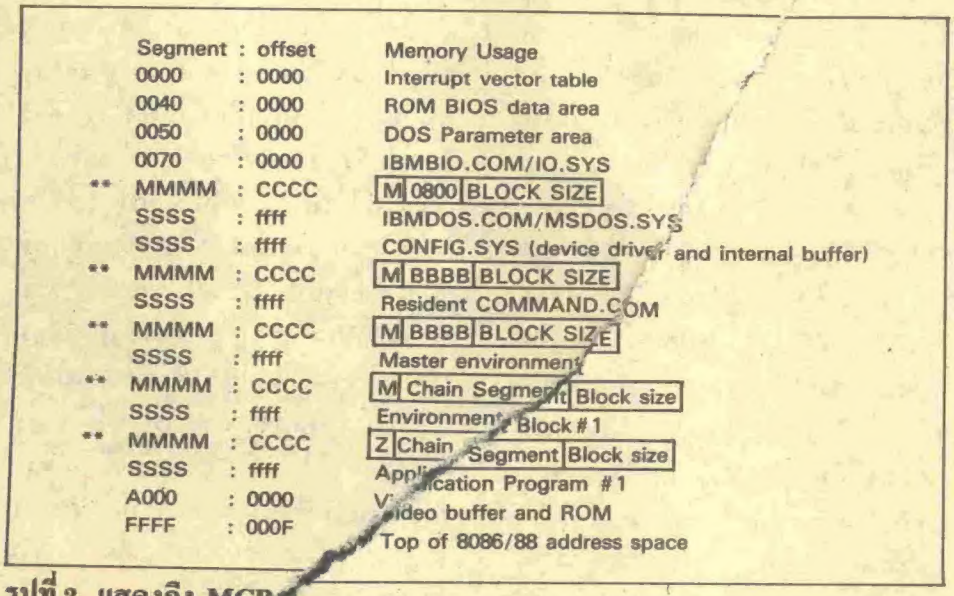
ก่อนที่จะเข้าสู่เนื้อเรื่องของหัวข้อนี้ เรามาทำความรู้จักกับ MCB ว่ามีหน้าตาอย่างไร

จากรูปที่ 1 คือ ส่วนที่เราเรียกว่า MCB ที่กล่าวไว้ในหัวข้อแรกมีขนาดเพียง 1 พารากราฟเท่านั้น จะประกอบด้วย 3 ส่วน และมีขนาด 1 ไบต์, 2 ไบต์ และ 2 ไบต์ส่วนที่เหลือไม่ใช้ ที่น่าสนใจก็คือไบต์ที่ 1-2 หากว่าเราดูรูปที่ 2 ประกอบ ตัวเลขที่ไบต์นี้เก็บไว้ จะพบว่าในบล็อกแรกนั้นจะต้องเป็น 0008H เสมอ ดังนั้นเราก็จะเดาได้ว่าในบล็อกแรกนั้นจะต้องเป็น IBMDOS.COM/MSDOS.SYS แน่นอน และที่ทำเครื่อง

หมาย ** ไว้เรียกว่าเมมโมรีคอนโทรลบล็อกหรือ MCB นั้นเอง ดังนั้นในบทความนี้จะเรียก 16 ไบต์นี้ว่า MCB ไบต์ที่ศูนย์หมายถึง Chain Segment และไบต์ที่ 3-4 จะหมายถึง Block Size

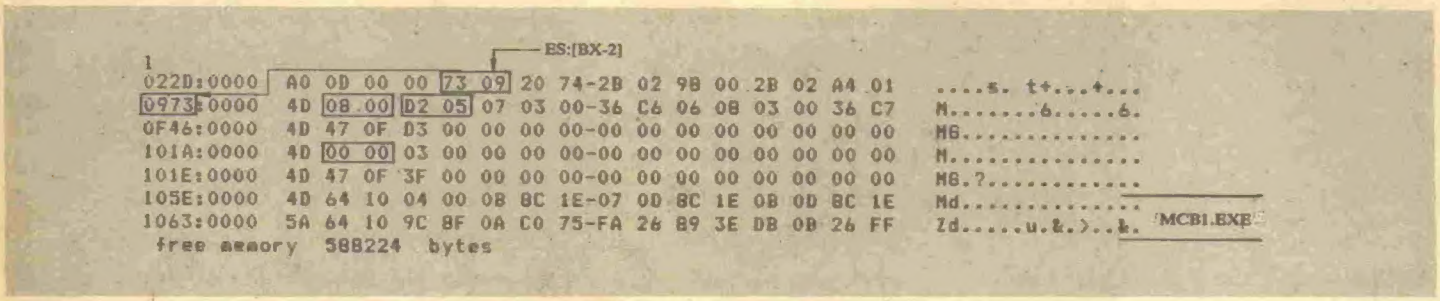
การเชื่อมโยงเป็นลูกโซ่ (Chain) ของ MCB นั้นจะเกิดขึ้นเมื่อโปรแกรมของเราร้องขอความจำเพิ่มจากที่มีอยู่ด้วยฟังก์ชัน 48H ดอสก็จะค้นหาขนาดของบล็อกความจำที่โตที่สุดให้ ถ้ามีเนื้อที่ความจำว่างเพียงพอ ดอสจะเปลี่ยน MCB นี้ให้รู้ว่าโปรแกรมใดเป็นเจ้าของ แต่ถ้าบล็อกที่พบนั้นโตเกินกว่าโปรแกรมร้องขอจากดอส MCB ของบล็อกก็จะเขียนขนาดของไบต์ที่ 3-4 ให้เล็กลงเท่ากับขนาดที่โปรแกรมต้องการ พร้อมกับสร้าง MCB ขึ้นใหม่ที่บริเวณหน่วยความจำที่ยังเหลืออยู่ จากนั้นก็จะทำการแก้ไข (update) พอยน์เตอร์ให้รู้ว่าเป็นลูกโซ่ (chain) หรือเขียนตัว "M" ที่ไบต์ที่ศูนย์ของ MCB นั้นเอง จากนั้นดอสจึงจะคืนคำตอบมาให้โปรแกรมว่าเซกเมนต์แอดเดรสของไบต์แรกที่จะใช้ได้อยู่ที่ใด เมื่อโปรแกรมคืนหน่วยความจำให้กับดอสด้วยฟังก์ชัน 49H ดอสก็มีหน้าที่เปลี่ยนแปลง MCB ให้ไบต์ที่ 1-2 นั้นเป็นว่างหรือ 0000 ไม่มีโปรแกรมใดเป็นเจ้าของเพื่อในการร้องขอให้หน่วยความจำในคราวต่อไป เมื่อโปรแกรมมีการเปลี่ยนแปลงขนาดหน่วยความจำด้วยฟังก์ชัน 4AH ให้มีขนาดเล็กลง ดอสก็จะสร้าง MCB ขึ้นเซกเมนต์ที่โปรแกรมเราไม่ใช้ พร้อมกับเขียนตัว "M" ไว้ที่ไบต์ศูนย์ พร้อมกับไบต์ที่ 1-2 และ 3-4 ให้รู้ ว่าเป็นลูกโซ่ต่อเนื่อง แต่ถ้าโปรแกรมร้องขอหน่วยความจำเพิ่มเติมจากดอสด้วยฟังก์ชัน 4AH ดอสก็จะเปลี่ยนแปลงไบต์ที่ 3-4 แทน และไม่บอกเซกเมนต์แอดเดรสของหน่วยความจำที่เพิ่มเติมให้ อย่างไรก็ตามต้องมีขนาดหน่วยความจำที่ยังว่างเหลืออยู่เพียงพอ มิฉะนั้นแล้วดอสก็จะบอกค่าผิดพลาดมาให้พร้อมกับปรับค่าไบต์ที่ 3-5 ให้มีขนาดใหม่ที่ถูกต้องตามที่โปรแกรมร้องขอ

รูปที่ 1 Memory Control Block (มีขนาด 1 พารากราฟ = 16 ไบต์)



รูปที่ 2 แสดงถึง MCB และ PC memory map

ถ้าดอสไม่สามารถจัดสรรขนาดความจำตามที่โปรแกรมต้องการได้เมื่อโปรแกรมร้องขอด้วยฟังก์ชัน 48H หรือ 4AH ดอสก็จะบอกค่าผิดพลาดให้กับแฟล็กตัวทด พร้อมกับรหัสผิดพลาดให้กับรีจิสเตอร์ AH ดังที่กล่าวมาในหัวข้อที่ 1 ลักษณะโปรแกรมที่ดีเมื่อทำงานครั้งแรกหลังจากโหลดเดอร์จัดการนำโปรแกรมลงในหน่วยความจำเสร็จเรียบร้อยแล้ว โปรแกรมนั้นควรจะเรียกฟังก์ชัน 4AH เพื่อลดขนาดความจำลงเท่าที่โปรแกรมทำงานได้ MCB บล็อกนี้จะอยู่ก่อน PSP การปล่อยให้หน่วยความจำว่างเพิ่มขึ้นเป็นวิธีที่ดีที่สุดของการออกแบบงานโปรแกรมหลายงานทำงานด้วยกัน (multitasking environments) เมื่อโปรแกรมจบการทำงาน คืนการทำงานให้กับดอส ดอสก็จะปล่อยให้หน่วยความจำว่าง โดยเขียนไว้ที่ MCB ไบต์ที่ 0, 1-2, และ 3-4 ตามลำดับ (MCB ที่โปรแกรมเป็นเจ้าของ) ก่อนที่จะคืนการทำงานไปให้ COMMAND.COM (หรือโปรแกรมที่เรียกใช้ด้วยดอสฟังก์ชัน 4BH โค้ดศูนย์) โปรแกรมที่เราเขียนขึ้นอาจเข้าไปจัดการ



รูปที่ 3 แสดงให้เห็นถึง MCB 6 บล็อก

หน่วยความจำได้เอง ทำให้ MCB บล็อกใดบล็อกหนึ่งเปลี่ยนแปลงไปหรือถูกทำลายลงโดยไม่ตั้งใจ และเป็นสาเหตุหนึ่งของการเกิดข้อผิดพลาดได้ ในคราวต่อไปเมื่อดอสตรวจสอบ MCB เพื่อจะจัดหน่วยความจำอีกครั้งมักจะตรวจพบก็จะเกิดข้อผิดพลาดโดยเขียนที่จอภาพ และหยุดการทำงานของคอมพิวเตอร์ไป ซึ่งเราจะต้องบูตเครื่องขึ้นใหม่ เราจะทราบได้อย่างไรว่า MCB บล็อกแรกอยู่ที่ส่วนใดของหน่วยความจำ ดอสมีฟังก์ชันการทำงานอยู่ (แต่ไม่เป็นที่เปิดเผย) นั่นคือ ฟังก์ชันหมายเลข 52H เมื่อเราเรียกดอสด้วยฟังก์ชันนี้แล้ว ค่าที่ได้จะอยู่ที่รีจิสเตอร์ ES:[BX-2] จะบอกว่าจุดเริ่มต้นของ MCB บล็อกแรก ดังรูปที่ 3 เป็นรูปที่ได้จากการรันโปรแกรมที่เขียนขึ้นเพื่อดู MCB ทุก ๆ บล็อกที่โยงถึงกัน

จากรูปที่ 3 นี้ ได้มาจากโปรแกรมแรกชื่อว่า MCB1.EXE ที่เขียนด้วยภาษาเทอร์โบเบสิก V.1.0 (รายละเอียดของโปรแกรมจะอยู่ในหัวข้อที่ 3) และพิมพ์ผลออกมาในรูปแบบของดีบัก (ที่เรารู้จักกันดี) เรามาดูบรรทัดแรกก่อน (ได้จากย่อหน้าที่แล้ว) โปรแกรมก็ทำการพิมพ์ข้อมูลที่อยู่พารากราฟนั้นออกมาดูทางหน้าจอ ดังนั้นเวิร์ดที่รีจิสเตอร์ ES : [BX-2] ชี้ ไปที่เซกเมนต์ของ MCB หมายเลข 1 ก็คือบรรทัดที่ 2 (ในรูปที่ 3) บรรทัดนี้ (ในที่นี้ 0973:0000 ในเครื่องบางรุ่นหรือดอสคนละเวอร์ชันจะไม่ตรงกัน) ไบต์ที่ศูนย์จะเป็นตัวอักษร "M" หมายถึง Chain ID เป็นรหัสที่บอกดอสว่ายังมี MCB ต่อไปอีก, ไบต์ที่ 1-2 มีค่าเป็น 08 00 มองเป็นเวิร์ดจะเป็น 0008 ก็หมายถึง MCB ของ IBMDOS.COM/MSDOS .SYS ขนาดความโตที่อยู่ภายใต้ความควบคุมของ MCB บล็อกนี้ ดูจากไบต์ที่ 3-4 ในที่นี้ D2 05 หากเป็นเวิร์ดก็คือ 05D2 แสดงว่า MCB บล็อกนี้ได้ควบคุมความจำไว้ 05D2H พารากราฟจากนั้นนำเอาแอดเดรสเซกเมนต์นี้ (คือ 0973H) บวกกับขนาดของ MCB (คือ 05D2H) แล้วบวกเพิ่มอีก 1 ก็คือ 0973H+05D2H+1 = 0F46 โปรแกรม MCB1.EXE ทำการคำนวณเรียบร้อยก็พิมพ์บรรทัดที่ 3 ออกมาไล่ไปเรื่อยแบบนี้จนถึง MCB บล็อกสุดท้ายคือบรรทัดที่ 7 (1063:0000) ไบต์ที่ศูนย์จะเป็นตัว "Z" หมายถึงบล็อกสุดท้ายนั่นเอง ส่วนบรรทัดที่ 4 (101A:0001) เป็นค่าเลข 00 00 2 ไบต์ติดกัน หมายถึง หน่วยความจำที่ว่างอยู่สำหรับดอสเวอร์ชัน 3.3 บริเวณนี้เป็นส่วนที่เปิดเครื่องครั้งแรก โปรแกรม AUTO EXEC.BAT จะถูกโหลดลงที่ตำแหน่งนี้ (ดูได้จากรูปที่ 10 จะมีเฉพาะดอส 3.3 ขึ้นไป) รูปที่ 4 นั้น ได้จากการทดลองโปรแกรมประเภทที่ฝังในหน่วยความจำหลาย ๆ ครั้ง จริง ๆ แล้วได้ทดลองหลาย ๆ รูปแบบ แต่ไม่สามารถจะทำภาพที่ได้มาลงทั้งหมดจึงตัดเอาเฉพาะส่วนที่จำเป็นมาประกอบการอธิบายเท่านั้นเอง ผู้อ่านอาจทดลองรันโปรแกรมเอง

ใช้โปรแกรมเทอร์โบเบสิกเข้าตรวจดูการจัดสรรหน่วยความจำ

โปรแกรมใช้ในบทความนี้มีอยู่ 4 โปรแกรมคือ TSR1.ASM ดังรูปที่ 5 (เป็นโปรแกรมเรสซิเดนต์) โปรแกรม MCB1.BAS, ดังรูปที่ 6 โปรแกรม MCB2.BAS ดังรูปที่ 7 และโปรแกรม MCB3.BAS ดังรูปที่ 8 ก่อนที่จะถึงเรื่องของโปรแกรม ดูโครงสร้างหน่วยความจำขณะที่รัน (run-time map) ของโปรแกรมที่ได้มาจากเทอร์โบเบสิกจากรูปที่ 9 การจัดเซกเมนต์ต่าง ๆ ภายในหน่วยความจำของโปรแกรมที่ได้จากการคอมพายล์ด้วยเทอร์โบเบสิกนั้นจะใช้หน่วยความจำที่เหลือทั้งหมด (แต่จะไม่ขอกล่าวและวิจารณ์ข้อดีข้อเสียของเทอร์โบเบสิก) เอาเป็นว่า ถ้าดูรูปที่ 3, รูปที่ 4 แล้วจะเห็น MCB บล็อกสุดท้ายที่มีชื่อเป็น "Z" ไบต์ที่ 3-4 จะมีขนาดโตที่สุด หากเรานำเวิร์ดนี้เปลี่ยนเป็นเลขฐาน 10 จะได้จำนวนไบต์ว่า MCB บล็อกสุดท้ายนี้ควบคุมพื้นที่หน่วยความจำเท่าไร (เป็นของโปรแกรมนั้น) ในรูปที่ 3 และ 4 จะบอกจำนวนไบต์ของบล็อกนี้ขนาดหน่วยความจำไม่เท่ากัน คือส่วนที่เหลือนั้นเอง

รูปที่ 4 MCB ที่ได้จากการรันโปรแกรมต่าง ตามลำดับดังนี้ TSR1.COM 2 ครั้ง COMMAND.COM /P 1 ครั้ง และ TSR1.COM 1 ครั้ง

```
022D:0000  A0 0D 00 00 73 09 20 74-2B 02 9B 00 2B 02 A4 01   ....s. t+...+...
0973:0000  4D 0B 00 D2 05 07 03 00-36 C6 06 0B 03 00 36 C7   M.......6.....6.
0F46:0000  4D 47 0F D3 00 00 00 00-00 00 00 00 00 00 00 00   MG..............
101A:0000  4D 00 00 03 00 00 00 00-00 00 00 00 00 00 00 00   M...............
101E:0000  4D 47 0F 3F 00 00 00 00-00 00 00 00 00 00 00 00   MG.?............
105E:0000  4D 64 10 04 00 0B 8C 1E-07 0D 8C 1E 0B 0D 8C 1E   Md..............
1063:0000  4D 64 10 10 00 0A C0 75-FA 26 B9 3E DB 0B 26 FF   Md.....u.&.>..&.
1074:0000  4D 7A 10 04 00 00 00 00-00 00 00 00 00 00 00 00   Mz..............
1079:0000  4D 7A 10 10 00 4D 43 42-29 0D 0A 74 68 69 73 20   Mz...MCB)..this
108A:0000  4D 91 10 05 00 00 00 00-00 00 00 00 00 00 00 00   M...............
1090:0000  4D 91 10 D3 00 9A F0 FE-1D F0 2F 01 47 0F 3C 01   M........./.G.<.
1164:0000  4D 91 10 0A 00 00 00 00-00 00 00 00 00 00 00 00   M...............
116F:0000  4D 75 11 04 00 00 00 00-00 00 00 00 00 00 00 00   Mu..............
1174:0000  4D 75 11 10 00 CD 21 3D-03 1E 74 12 BA C2 15 B4   Mu....!=..t.....
1185:0000  4D BB 11 04 00 00 00 00-00 00 00 00 00 00 00 00   M...............
118A:0000  5A BB 11 75 BE 4D 43 42-29 0D 0A 74 68 69 73 20   Z..u.MCB)..this
 free memory  583504  bytes
```

TSR1.EXE

TSR1.EXE

COMMAND.COM /P

TSR1.EXE

MCB1.EXE

รูปที่ 5 โปรแกรม TSR1.ASM

```
 1 dosseg
 2 .model small
 3 .data
 4 say1 db 'Program show DOS memory control block (MCB)',13,10
 5      db 'this program is terminate but resident',13,10
 6      db 'Run this program first or many times',13,10
 7      db 'then you can run other programs such as : '
 8      db  'MCB1.EXE, MCB2.EXE, MCB3.EXE'
 9 say1_len equ $-say1
10 .stack  16
11 .code
12 start: mov ax, @data
13        mov ds, ax
14        mov dx, offset say1
15        mov cx, say1_len
16        mov bx, 1
17        mov ah, 40h
18        int 21h
19        mov dx, ss
20        mov ax, cs
21        sub dx, ax
22        inc dx
23        mov ah,31h
24        int 21h
25        end start
```

รูปที่ 6 โปรแกรม MCB1.BAS

```
1 %AX = 1                                                'set constant register name
2 %BX = 2
3 %ES = 9
4 '-------------- Main program ------------
5 cls                                                    'clear screen
6 REG %AX, &h5200                                        'call dos function 52h
7 call interrupt &h21                                    'undocumented
8 mcb.seg% = (REG (%ES)): mcb.off% = REG (%BX) - 2       'set pointer ES:[BX-2]
9                                                        'to point MCB segment
10 mcb.seg% = mcb.seg% + (mcb.off%/16)                   'make to paragraph
```

รูปที่ 6 (ต่อ)

```
11 r%=1                                                 'first paragrap
12 call seemcb(mcb.seg%,r%)                             'see first ES:[BX-2]
13  def seg = mcb.seg%                                  'get first MCB
14 mcb.seg% = (peek(mcb.off% and 7)) + (256 * peek((mcb.off% and 7) + 1))
15 def seg
16 call seemcb(mcb.seg%,r%)                             'see secound MCB
17 do
18    call chain.mcb(mcb.seg%,chain.id%,chain.seg%,block.size%)
19    if chr$(chain.id%) = "Z" then exit loop           'do until ID = 'Z'
20    mcb.seg% = mcb.seg% + block.size% + 1
21    call seemcb(mcb.seg%,r%)
22 loop
23
24 block.size$ = hex$(block.size%)
25 memory& = 0 : position% = 0
26 call convert.hex.dec(position%,memory&,block.size$)
27 print " free memory "memory&" bytes"
28 end   'End main program
29
30 '--------- sub program convert Hex to -------------
31 '         long interger whith recursive
32 sub convert.hex.dec(position%,memory&,block.size$)
33    if position% > len(block.size$) then
34       exit sub
35    end if
36    memory& = (memory& * 16) + val("&h"+mid$(block.size$,position% + 1,1))
37    incr position%
38    call convert.hex.dec(position%,memory&,block.size$)
39 end sub
40
41 '--------- sub program to find chain MCB ----------'
42 'segment:offset byte  #0     #1-2      #3-4          '
43 '                  chain  parent    block          '
44 '                    ID     PID      size           '
45 '-------------------------------------------------'
46 sub chain.mcb(mcb.seg%,chain.id%,chain.seg%,block.size%)
47   def seg       = mcb.seg%                           'segment of chain MCB
48     chain.id%   = peek(0)                            'byte #0 = chain ID
49     chain.seg%  = peek(1) + (256 * peek(2))          'word #1 = parent PID
50     block.size% = peek(3) + (256 * peek(4))          'word #2 = block size
51   def seg
52 end sub
53
54 '--------- sub program to print out data --------
55 '          in fromat debug
56 sub seemcb(mcb.seg%,r%)
57 local c1%,c2%,y$,mcb%,mcb.off%
58       c1%=12 : c2%=64 : mcb.off%=0
59  y$="0000" + hex$(mcb.seg%)+":" : y$=right$(y$,5)
60  locate r%,1,1 : print y$; : print right$("0000"+hex$(mcb.off%),4)
61   for offset%=0 to 15
62     def seg = mcb.seg%
63        mcb% = peek(offset%)
64     def seg
65     y$=hex$(mcb%) : y$="0"+y$ : y$=right$(y$,2)
66     locate r%,c1%,1 : print y$
67     if c1%=36 then locate r%,35,1 : print "-"
68     if mcb%>127 or (mcb%<32) then mcb%=asc(".")
69     locate r%,c2%,1 : print chr$(mcb%)
70     incr c1%,3 : incr c2%
71  next offset%
72  incr r%
73  if r%>23 then        'display one screen 23 line
74     a$=input$(1)
75     cls
76     r%=1              'clear to start row
77  end if
78 end sub
```

รูปที่ 7 โปรแกรม MCB2.BAS

```
%AX = 1                                                 'set constant register name
%BX = 2
%ES = 9
'--------------- Main program ------------
cls                                                     'clear screen
REG %AX, &h5200                                         'call dos function 52h
call interrupt &h21                                     'undocumented
mcb.seg% = (REG (%ES)): mcb.off% = REG (%BX) - 2        'set pointer ES:[BX-2]
                                                        'to point MCB segment
mcb.seg% = mcb.seg% + (mcb.off%/16)                     'make to paragraph
r%=1                                                    'first paragrap
call seemcb(mcb.seg%,r%)                                'see first ES:[BX-2]
 def seg = mcb.seg%                                     'get first MCB
mcb.seg% = (peek(mcb.off% and 7)) + (256 * peek((mcb.off% and 7) + 1))
def seg
call seemcb(mcb.seg%,r%)                                'see secound MCB
do
   call chain.mcb(mcb.seg%,chain.id%,chain.seg%,block.size%)
   if chr$(chain.id%) = "Z" then exit loop              'do until ID = 'Z'
   mcb.seg% = mcb.seg% + block.size% + 1                'calculete next chain MCB
   call seemcb(mcb.seg%,r%)
   envr.seg% = mcb.seg% + 1
   def seg   = envr.seg%
      psp%   = peek(0) + (256 * peek(1))
   def seg
   if psp% = &h20cd then
      call psp.seg(envr.seg%,abc.off%,r%)
   else
      select case psp%
        case 0                                          'free memory control block
          exit select
        case 8                                          'IBMDOS.COM/MSDOS.SYS
          exit select
        case else                                       'chain MCB
          call environment(envr.seg%,block.size%,r%)
      end select
   end if
loop
block.size$ = hex$(block.size%)
memory& = 0 : position% = 0
call convert.hex.dec(position%,memory&,block.size$)
print " free memory "memory&" bytes"
end  'End main program


sub convert.hex.dec(position%,memory&,block.size$)
   if position% > len(block.size$) then exit sub
   memory& = (memory& * 16) + val("&h"+mid$(block.size$,position% + 1,1))
   incr position%
   call convert.hex.dec(position%,memory&,block.size$)
end sub


sub chain.mcb(mcb.seg%,chain.id%,chain.seg%,block.size%)
  def seg      = mcb.seg%
  chain.id%    = peek(0)
  chain.seg%   = peek(1) + (256 * peek(2))
  block.size%  = peek(3) + (256 * peek(4))
  def seg
end sub


sub seemcb(mcb.seg%,r%)
local y$, a$, mcb%
local c1%,     c2%,      mcb.off%
      c1%=12 : c2%=64  : mcb.off%=0
 y$="0000" + hex$(mcb.seg%)+":" : y$=right$(y$,5)
 if r%>23 then          'display one screen 23 line
    a$=input$(1)
```

รูปที่ 7 (ต่อ)

```
    cls
    r%=1
  end if
  locate r%,1,1 : print y$; : print right$("0000"+hex$(mcb.off%),4)
   for offset%=0 to 15
     def seg = mcb.seg%
        mcb% = peek(offset%)
     def seg
     y$=hex$(mcb%)    : y$="0"+y$    : y$=right$(y$,2)
     locate r%,c1%,1 : print y$
     if c1%=36 then locate r%,35,1 : print "-"
     if mcb%>127 or (mcb%<32) then mcb%=asc(".")
     locate r%,c2%,1 : print chr$(mcb%)
     incr c1%,3      : incr c2%
  next offset%
  incr r%
end sub

'--------- sub prograa to print out environaent segaent -------
sub environaent(envr.seg%,tap.mcb.off%,r%)
local tap.mcb.seg%, tap.off%
local c1%, c2%, y$, mcb%, a$, flag%, f%
tap.mcb.seg%=envr.seg% : mcb.off%=0
while (flag%=0)
   c1%=12 : c2%=64
   y$="0000" + hex$(tap.mcb.seg%)+":" : y$=right$(y$,5)
    if r%>23 then        'display one screen 23 line
     a$=input$(1)
     cls
     r%=1
  end if
   locate r%,1,1 : print y$; : print right$("0000"+hex$(mcb.off%),4)
   for offset%=mcb.off% to mcb.off%+15
     def seg = tap.mcb.seg%
        mcb% = peek(offset%)
     if mcb% = 0 then f% = peek(offset%+1)
     def seg
     if mcb% = 0 and f%=0 then
        flag%   =1
        tap.off%=offset% + 2
     end if
     y$=hex$(mcb%)    : y$="0"+y$ : y$=right$(y$,2)
     locate r%,c1%,1 : print y$
     if c1%=36 then
        locate r%,35,1
        print "-"
     end if
     if mcb%>127 or (mcb%<32) then mcb%=asc(".")
     locate r%,c2%,1 : print chr$(mcb%)
     incr c1%,3      : incr c2%
  next offset%
   incr r%
  incr mcb.off%,16
wend
def seg = tap.mcb.seg%
    mcb% = peek(tap.off%)
def seg
if mcb% = 1 then incr tap.off%,2
while (flag%=1)
   c1%=12 : c2%=64
   y$="0000" + hex$(tap.mcb.seg%)+":" : y$=right$(y$,5)
    if r%>23 then        'display one screen 23 line
     a$=input$(1)
     cls
     r%=1
    end if
   locate r%,1,1 : print y$; : print right$("0000"+hex$(mcb.off%),4)
```

รูปที่ 7 (ต่อ)

```
  for offset%=mcb.off% to mcb.off%+15
    def seg = tmp.mcb.seg%
       mcb% = peek(offset%)
    def seg
    if mcb%=0 then flag%=0
    y$=hex$(mcb%) : y$="0"+y$ : y$=right$(y$,2)
    locate r%,c1%,1 : print y$
    if c1%=36 then
       locate r%,35,1
       print "-"
    end if
    if mcb%>127 or (mcb%<32) then mcb%=asc(".")
    locate r%,c2%,1 : print chr$(mcb%)
    incr c1%,3      : incr c2%
  next offset%
   incr r%
   incr mcb.off%,16
 wend
 end sub

 '---------- sub program to print out PSP segment --------
 sub psp.seg(mcb.seg%,envr.seg%,r%)
 local mcb.off%, c1%, c2%, y$, mcb%, a$
      ' mcb.off%=0
 while (mcb.off% < &h80)
  c1%=12 : c2%=64
  y$="0000" + hex$(mcb.seg%)+":" : y$=right$(y$,5)
  if r%>23 then          display one screen 23 line
     a$=input$(1)
     cls
     r%=1
  end if
  locate r%,1,1 : print y$; : print right$("0000"+hex$(mcb.off%),4)
   for offset%=mcb.off% to mcb.off%+15
     def seg = mcb.seg%
        mcb% = peek(offset%)
     def seg
     y$=hex$(mcb%) : y$="0"+y$ : y$=right$(y$,2)
     locate r%,c1%,1 : print y$
     if c1%=36 then locate r%,35,1 : print "-"
     if mcb%>127 or (mcb%<32) then mcb%=asc(".")
     locate r%,c2%,1 : print chr$(mcb%)
     incr c1%,3 : incr c2%
  next offset%
  incr r%
  incr mcb.off%,16
 wend
 def seg      = mcb.seg%
   envr.seg% = peek(&h2c) + (256 * peek(&h2d)) 'return value psp:2ch
 def seg
 end sub
```

รูปที่ 8 โปรแกรม MCB3.BAS

```
 1 %AX = 1                                          set constant register name
 2 %BX = 2
 3 %ES = 9
 4
 5 '--------------- Main program ------------
 6 cls                                               clear screen
 7 REG %AX, &h5200                                   call dos function 52h
 8 call interrupt &h21                               undocumented
 9 mcb.seg% = (REG (%ES)): mcb.off% = REG (%BX) - 2  set pointer ES:[BX-2]
10                                                   to point MCB segment
```

รูปที่ 8 (ต่อ)

```
11 mcb.seg% = mcb.seg% + (mcb.off%\16)              'make to paragraph
12 par.psp%  = mcb.seg%
13 def seg   = mcb.seg%                                get first MCB
14   mcb.seg% = (peek(mcb.off% and 7)) + (256 * peek((mcb.off% and 7) + 1))
15 def seg
16
17 call header
18 call mcb(mcb.seg%)
19 call prn.mcb(mcb.seg%)
20 do until chr$(chain.id%)="Z"
21   mcb.seg% = mcb.seg% + mcb.size% +1
22   call mcb(mcb.seg%)
23   call prn.mcb(mcb.seg%)
24 loop
25 print string$(70,"=")
26 end
27
28 sub header
29   print string$(70,"=")
30   print "MCB  MCB CHAIN CHAIN   BLOCK PARENT  ENV    OWNER
31   print "NO.  SEG  ID.   SEG    SIZE   PSP.  BLOCK
32   print string$(70,"=")
33 end sub
34
35
36 sub mcb(mcb.seg%)
37 shared chain.id%, chain.seg.mcb%, mcb.size%
38   def seg = mcb.seg%
39    chain.id% = peek(0)
40    chain.seg.mcb%   = peek(1) + (256 * peek(2))
41    mcb.size% = peek(3) + (256 * peek(4))
42   def seg
43 end sub
44
45
46 sub prn.mcb(mcb.seg%)
47 shared chain.id%, chain.seg.mcb%, mcb.size%
48 static cnt% = 0
49 local parid, mcbseg%, envf$, envr.seg%
50 incr cnt%
51 def seg = chain.seg.mcb%
52  parid% = peek(&h16) + (256 * peek(&h16 + 1))
53  envr.seg% = peek(&H2c) + (256 * peek(&h2c + 1))
54 def seg
55 mcbseg% = mcb.seg%
56 if (mcbseg% + 1) = envr.seg% then
57    envf$ = "Y"
58 else
59    envf$ = "N"
60 end if
61 block.size$ = hex$(mcb.size%)
62 memory& = 0 : position% = 0
63 call convert.hex.dec(position%,memory&,block.size$)
64 print using "##";cnt%;
65 print using "  \  \";right$("0000"+hex$(mcb.seg%),4);
66 print using "    !";chr$(chain.id%);
67 print using "     \  \";right$("0000"+hex$(chain.seg.mcb%),4);
68 print using "  ######";memory&;
69 print using "  \  \";right$("0000"+hex$(parid%),4);
70 print using "     !     ";envf$;
71 call prn.pid.own(chain.seg.mcb%, parid%)
72 end sub
73
74
75 sub convert.hex.dec(position%,memory&,block.size$)
76    if position% > len(block.size$) then exit sub
77    memory& = (memory& * 16) + val("&h"+mid$(block.size$,position% + 1,1))
78  ' incr position%
```

รูปที่ 8 (ต่อ)

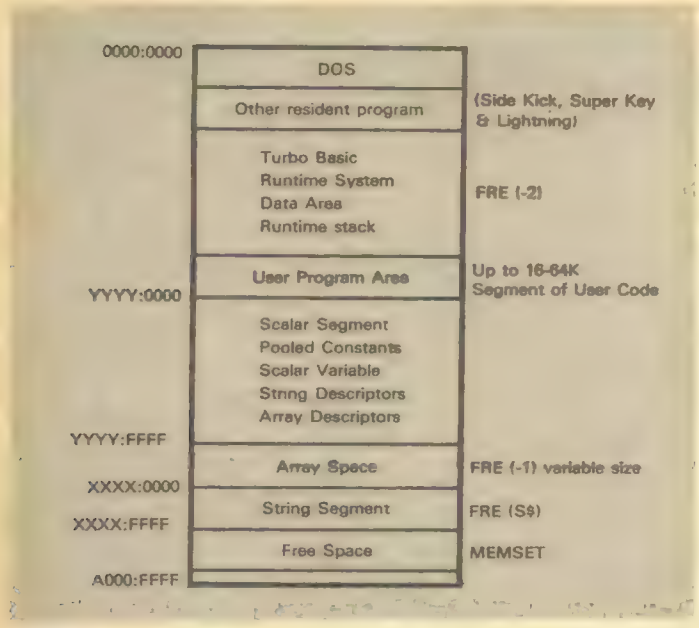
```
    call convert.hex.dec(position%,memory&,block.size$)
end sub


sub prn.pid.own(chain.seg.mcb%, parent%)
local envsegptr%, envptr$, envsizeptr%, envsize%, x% = 0
static ccnum% = 1
static prev.chain.seg.mcb% = &hffff
select case chain.seg.mcb%
    case 0
      print "FREE MEMORY CONTROL BLOCK"
      exit SUB
    case 8
      print "IBMDOS.COM/MSDOS.SYS"
      exit sub
end select

def seg = chain.seg.mcb%
  envsegptr% = peek(&h2c) + (256 * peek(&h2c + 1))
def seg
def seg = envsegptr%
  envptr%= peek(0) + (256 * peek(1))
def seg

def seg = envsegptr% - 1
  envsizeptr% = peek(3) + (256 * peek(4))
def seg

envsize% = envsizeptr% * 16
if (chain.seg.mcb% = parent%) then
   if prev.pid% <> chain.seg.mcb% then
       incr ccnum%
   end if
   print "COMMAND.COM COPY ##";
   print using "##";ccnum%
   prev.pid% = chain.seg.mcb%
   exit sub
end if
while (envsize% > 0)
  x% = 1
  while (x% <> 0)
    decr envsize%
    def seg = envsegptr%
         x% = peek(envsize%)
    def seg
  wend
  decr envsize%
  def seg = envsegptr%
      x%  = peek(envsize%)
  def seg
  if x% = 1 then
    incr envsize%,2
    exit loop
  end if
wend
if envsize%=0 then
   beep
   exit sub
end if
x%=1
while (x% <> 0)
   def seg = envsegptr%
       x%  = peek(envsize%)
   def seg
   print chr$(x%);
   incr envsize%
wend
end sub
```

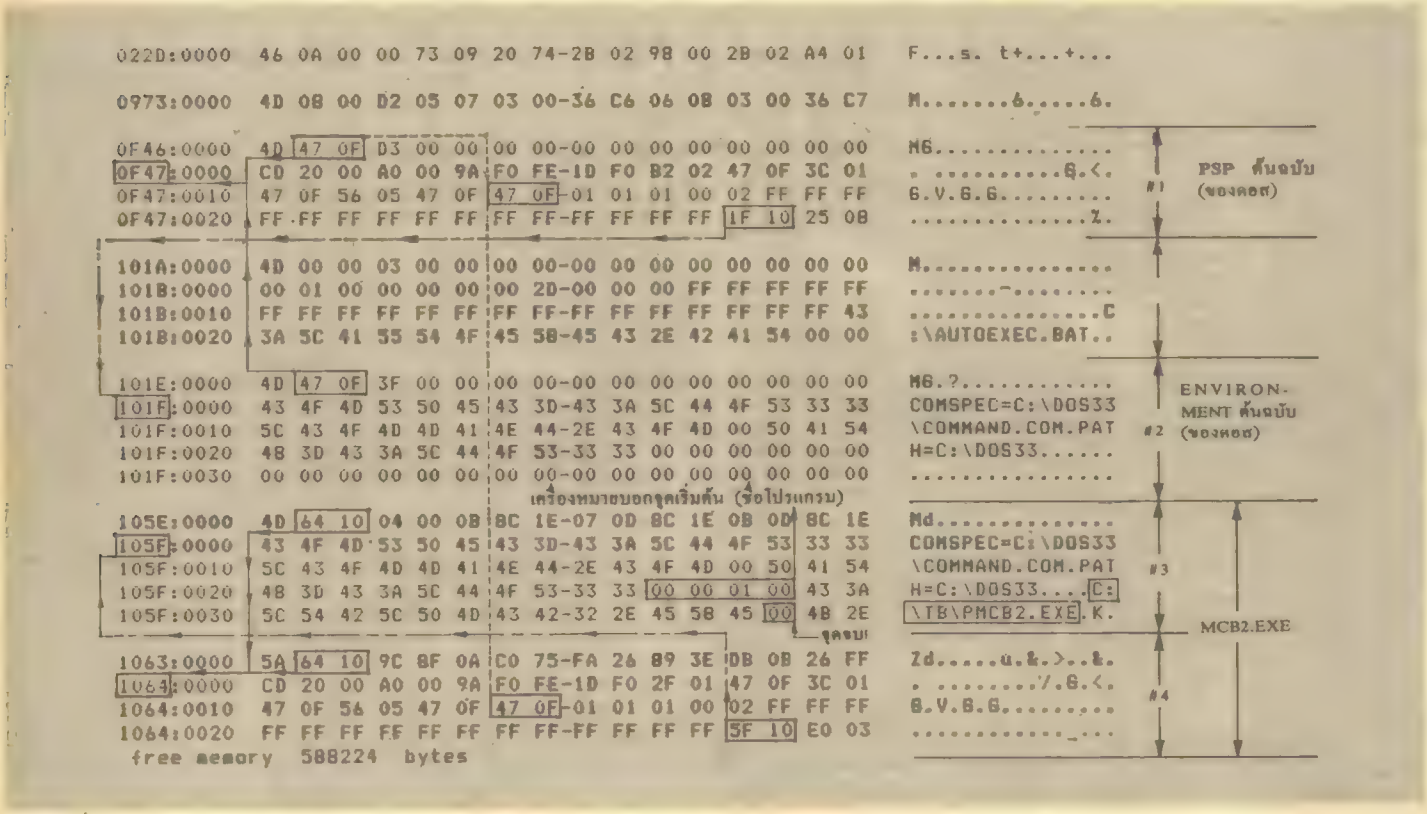


รูปที่ 9 Run-time Map ของเทอร์โบเบสิก

หลักการของโปรแกรม

โปรแกรม MCB1.BAS มีหลักการสรุปได้ 4 ข้อใหญ่ ๆ คือ

การทำงานของโปรแกรม MCB1.BAS	บรรทัดที่
1. เรียกใช้ฟังก์ชัน 52H ผ่านทาง int 21H	1-7
2. ดูหน่วยความจำตรงตำแหน่ง ES:[BX-2]	8-16
3. ลูปจนกว่าจะพบไบต์ที่ 0 = "Z"	17-22
3.1 ดู MCB บล็อกนั้นไบต์ที่ 0, 1-2, 3-4	18
3.2 คำนวณหาลูกโซ่หน่วยความจำ	20
3.3 พิมพ์รายละเอียดของ MCB บล็อกนั้น	21
4. นำไบต์ที่ 3-4 นั้นมาเปลี่ยนเป็นเลขฐาน 10	24-28



รูปที่ 10 แสดงให้เห็นพื้นที่หน่วยความจำที่เป็นเจ้าของ MCB แต่ละบล็อก

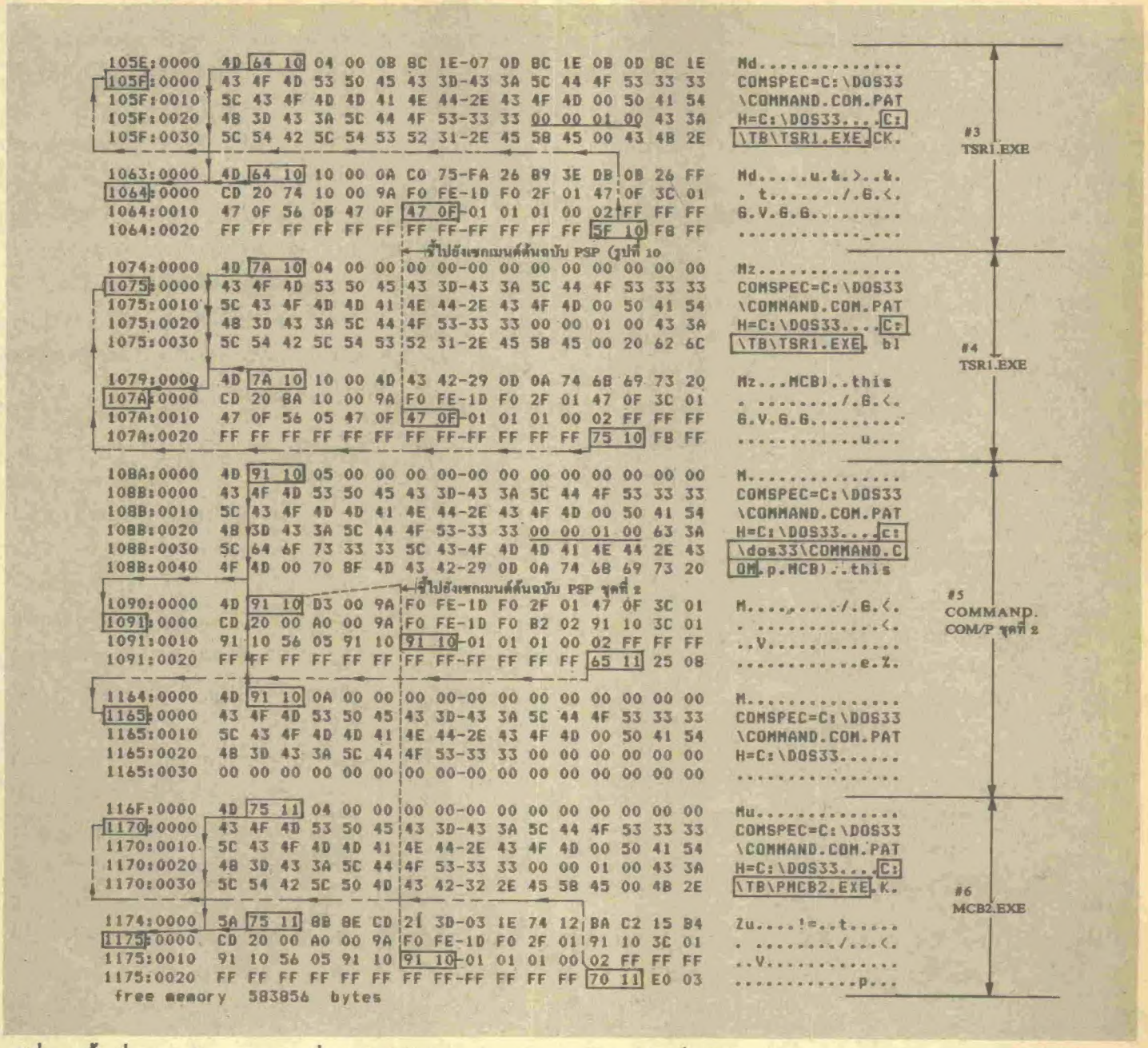
#1 ต้นฉบับ PSP ที่ COMMAND.COM สร้างขึ้น

#2 ต้นแบบของ Environment บล็อก

#3 Environment บล็อกของโปรแกรม MCB2.EXE

#4 PSP และเนื้อที่ของโปรแกรม MCB2.EXE

จะแสดงให้เห็นเพียง 3 พารากราฟเท่านั้น



รูปที่ 11 พื้นที่ของหน่วยความจำที่อยู่ภายใต้การควบคุมของ MCB ส่วนเริ่มต้นของ MCB#1 และ #2 เหมือนกับรูปที่ 10

เมื่อรันโปรแกรมนี้แล้วจะได้ภาพที่หน้าจอดังรูปที่ 3 และรูปที่ 4 ส่วนรายละเอียดอื่นของโปรแกรมขออนุญาตไม่กล่าวถึง

โปรแกรม MCB2.BAS เป็นโปรแกรมที่ปรับรูปแบบต่อจากโปรแกรม MCB1.BAS หลักการนั้นก็ยังคงเดิม แต่เพิ่มในส่วนของโปรแกรมให้พิมพ์พื้นที่ของหน่วยความจำ ที่อยู่ภายใต้การดูแลของแต่ละ MCB บล็อกออกมาเพียง 3 บรรทัด บรรทัดที่ 23-37 ของโปรแกรมทำให้ได้รูปที่ 10 และ 11

จากรูปที่ 10 และรูปที่ 11 จะเป็นภาพที่ต่อเนื่องกันได้ใช้เครื่องหมาย # และตัวเลขกำกับไว้ที่บล็อกพื้นที่หน่วยความจำ คุณสมบัติของ PSP (Program Segment Prefix) มีรายละเอียดเพิ่มเติมอยู่ 3 ประการคือ

1. PSP:0016 ขนาด 2 ไบต์ จะบอกว่า PSP ต้นฉบับนั้นดอสได้มาจากเซกเมนต์ไหน

2. PSP:002C ขนาด 2 ไบต์ จะบอกว่า Environment ของโปรแกรมอยู่ที่ไหน

3. ไบต์ต่อเนื่องที่มีลักษณะ 00 00 01 00 แล้วต่อจากนั้นจะเป็นชื่อของโปรแกรมที่เป็นเจ้าของพื้นที่หน่วยความจำบล็อกเหล่านี้ (ได้มาจากผลของข้อ 2)

ในรูปที่ 10 นั้นเมื่อเราเปิดเครื่องใหม่ ๆ แล้วใช้โปรแกรม MCB2.EXE เข้าตรวจดู MCB ของระบบ และรูปที่ 11 ได้จากการรันโปรแกรมต่าง ๆ ดังนี้

รันโปรแกรม TSR1.EXE 2 ครั้ง ฝังในหน่วยความจำ (#3 และ #4)

รัน COMMAND.COM /P 1 ครั้ง (#5)

รันโปรแกรม MCB2.EXE ตรวจดูผลของ MCB เปลี่ยนแปลงไปอย่างไร (# 6) ทั้งรูปที่ 10 และ 11 ใช้เส้นที่มีความหมายดังนี้

——— หมายถึง chain PSP

----- หมายถึงเซกเมนต์แอดเดรสของ PSP ต้นฉบับ

———— หมายถึงเซกเมนต์แอดเดรสของ ENVIRONMENT

จากคุณสมบัติของ PSP 3 ประการ และความหมายของเส้น คิดว่าแฟน ๆ ไมโครคงจะมองออกนะครับ (ใจเย็น ๆ ไล่ดูตามทิศของลูกศร) ได้นำเอาโปรแกรม MCB2.BAS มาเขียนใหม่หมดเป็นโปรแกรม MCB3.BAS โดยมีหลักการดังนี้

การทำงานของโปรแกรม MCB3.BAS	บรรทัดที่
1. เรียกใช้ดอสฟังก์ชัน 52H	1-8
2. ดูไปที่ MCB บล็อกแรก	9-15
3. พิมพ์หัว (Header) ของตาราง	17
4. พิมพ์ MCB บล็อกแรก	18-19
5. ลูปจนกระทั่งพบ ID = "Z"	20-24
5.1 คำนวณหา MCB บล็อกลูกโซ่ต่อไป	
5.2 ดูค่าไบต์ที่ 0, 1-2 และ 3—4	
5.3 พิมพ์ MCB บล็อกต่อไป	
6. พิมพ์เส้นปิดท้าย (End line)	25

จากการทดลองเปลี่ยนแปลงไฟล์ CONFIG.SYS โดยปรับขนาดของบัฟเฟอร์บ้าง เพิ่มอุปกรณ์เข้าไปในระบบ (รูปที่ 12-14) จะพบการเปลี่ยนแปลงของ MCB บล็อกที่ IBMDOS.COM/MSDOS.SYS ครอบครอง โดยที่ขนาดโตขึ้นตามจำนวนอุปกรณ์ที่เราติดตั้งในระบบ (ต้องบูตเครื่องใหม่ทุกครั้งที่เราแก้ไขไฟล์ (CONFIG.SYS)

จากรูปที่ 15-18 เมื่อเราทดลองฝังโปรแกรมต่าง ๆ ในหน่วยความจำแล้วใช้โปรแกรม MCB3.EXE เข้าตรวจสอบหน่วยความจำ (MCB) จะพบทุก ๆ โปรแกรมก็นับว่าโปรแกรม MCB3.EXE เป็นเครื่องมืออย่างหนึ่งของเราในการค้นหาชื่อโปรแกรมที่ฝังอยู่ในหน่วยความจำ

สรุป

จากจุดเริ่มต้นบล็อกแรกของ MCB ที่หาได้จากดอสฟังก์ชัน 52H จะได้ ES:[BX-2] เราสามารถที่จะหาบล็อกลูกโซ่ (chain) ไปโดยเอาไบต์ที่ 3-4 มาบวกกับเซกเมนต์แอดเดรสของมันเอง บวกเพิ่มอีก 1 พารากราฟ จนกว่าจะพบ ID = "Z" เป็นบล็อกสุดท้าย การจัดสรรหน่วยความจำควรจะเรียกผ่านดอสฟังก์ชัน 48H, 49H, 4AH หากสงสัยว่ามีโปรแกรมอะไรบ้างที่ฝังอยู่ในหน่วยความจำ เราสามารถที่ตรวจสอบได้ค้นหาชื่อของโปรแกรมที่เป็นเจ้าของพื้นที่หน่วยความจำนั้นได้ ดังเช่น โปรแกรม MCB2.BAS และ MCB3.BAS ลักษณะของ MCB นั้นเป็นลิงค์ลิสแบบเดินหน้า ไม่สามารถที่จะค้นหาย้อนกลับได้

```
=============================================================================
MCB   MCB CHAIN CHAIN    BLOCK PARENT  ENV     OWNER
NO.   SEG  ID.   SEG      SIZE   PSP.  BLOCK
=============================================================================
 1   0973   M    0008    13072   022B     N     IBMDOS.COM/MSDOS.SYS
 2   0CA5   M    0CA6     3376   0CA6     N     COMMAND.COM COPY ## 1
 3   0D79   M    0000       48   F000     N     FREE MEMORY CONTROL BLOCK
 4   0D7D   M    0CA6     1008   0CA6     Y     COMMAND.COM COPY ## 1
 5   0DBD   M    0DC3       64   0CA6     Y     C:\TB\MCB3.EXE
 6   0DC2   Z    0DC3   598992   0CA6     N     C:\TB\MCB3.EXE
=============================================================================
```

รูปที่ 12 เมื่อใช้ CONFIG.SYS ดังภาพ (Y หมายถึง ENVIRONMENT BLOCK)

```
buffers = 40
files = 40
break = on
device = c:\dos33\ansi.sys
shell = c:\dos33\command.com c:\dos33 /p /e:1000

=============================================================================
MCB   MCB CHAIN CHAIN    BLOCK PARENT  ENV     OWNER
NO.   SEG  ID.   SEG     SIZE   PSP.   BLOCK
=============================================================================
 1  0973    M     0008   24688  022B     N     IBMDOS.COM/MSDOS.SYS
 2  0F7B    M     0F7C    3376  0F7C     N     COMMAND.COM COPY ## 1
 3  104F    M     0000      48  F000     N     FREE MEMORY CONTROL BLOCK
 4  1053    M     0F7C    1008  0F7C     Y     COMMAND.COM COPY ## 1
 5  1093    M     1099      64  0F7C     Y     C:\tb\MCB3.EXE
 6  1098    Z     1099  587376  0F7C     N     C:\tb\MCB3.EXE
=============================================================================
```

รูปที่ 13 เมื่อใช้บัฟเฟอร์โตขึ้น MCB ของ IBMDOS.COM/MSDOS.SYS โตขึ้น

```
buffers = 40
files = 40
break = on
device = c:\dos33\ansi.sys
device = c:\dos33\vdisk.sys
shell = c:\dos33\command.com c:\dos33 /p /e:1000

=============================================================================
MCB   MCB CHAIN CHAIN    BLOCK PARENT  ENV     OWNER
NO.   SEG  ID.   SEG     SIZE   PSP.   BLOCK
=============================================================================
 1  0973    M     0008   90992  022B     N     IBMDOS.COM/MSDOS.SYS
 2  1FAB    M     1FAC    3376  1FAC     N     COMMAND.COM COPY ## 1
 3  207F    M     0000      48  F000     N     FREE MEMORY CONTROL BLOCK
 4  2083    M     1FAC    1008  1FAC     Y     COMMAND.COM COPY ## 1
 5  20C3    M     20C9      64  1FAC     Y     C:\tb\MCB3.EXE
 6  20CB    Z     20C9  521072  1FAC     N     C:\tb\MCB3.EXE
=============================================================================
```

รูปที่ 14 เมื่อเพิ่มอุปกรณ์แรมดิสค์

```
=============================================================================
MCB   MCB CHAIN CHAIN    BLOCK PARENT  ENV     OWNER
NO.   SEG  ID.   SEG     SIZE   PSP.   BLOCK
=============================================================================
 1  0973    M     0008   23840  022B     N     IBMDOS.COM/MSDOS.SYS
 2  0F46    M     0F47    3376  0F47     N     COMMAND.COM COPY ## 1
 3  101A    M     0000      48  F000     N     FREE MEMORY CONTROL BLOCK
 4  101E    M     0F47    1008  0F47     Y     COMMAND.COM COPY ## 1
 5  105E    M     1064      64  0F47     Y     C:\tb\TSR1.EXE
 6  1063    M     1064     256  0F47     N     C:\tb\TSR1.EXE
 7  1074    M     107A      64  0F47     Y     C:\tb\MCB3.EXE
 8  1079    Z     107A  587872  0F47     N     C:\tb\MCB3.EXE
=============================================================================
```

รูปที่ 15 การค้นหาโปรแกรมที่ฝังในหน่วยความจำ

```
=============================================================================
MCB   MCB CHAIN CHAIN    BLOCK PARENT  ENV     OWNER
NO.   SEG  ID.   SEG     SIZE   PSP.   BLOCK
=============================================================================
 1   0973   M    0008    23840  022B     N     IBMDOS.COM/MSDOS.SYS
 2   0F46   M    0F47     3376  0F47     N     COMMAND.COM COPY ## 1
 3   101A   M    0000       48  F000     N     FREE MEMORY CONTROL BLOCK
 4   101E   M    0F47     1008  0F47     Y     COMMAND.COM COPY ## 1
 5   105E   M    1064       64  0F47     Y     C:\tb\TSR1.EXE
 6   1063   M    1064      256  0F47     N     C:\tb\TSR1.EXE
 7   1074   M    107B       80  0F47     Y     C:\DOS33\GRAFTABL.COM
 8   107A   M    107B     1344  0F47     N     C:\DOS33\GRAFTABL.COM
 9   10CF   M    10D5       64  0F47     Y     C:\tb\MCB3.EXE
10   10D4   Z    10D5   586416  0F47     N     C:\tb\MCB3.EXE
=============================================================================
```

รูปที่ 16 มีโปรแกรมฝังหน่วยความจำ 2 โปรแกรม

```
=============================================================================
MCB   MCB CHAIN CHAIN    BLOCK PARENT  ENV     OWNER
NO.   SEG  ID.   SEG     SIZE   PSP.   BLOCK
=============================================================================
 1   0973   M    0008    23840  022B     N     IBMDOS.COM/MSDOS.SYS
 2   0F46   M    0F47     3376  0F47     N     COMMAND.COM COPY ## 1
 3   101A   M    0000       48  F000     N     FREE MEMORY CONTROL BLOCK
 4   101E   M    0F47     1008  0F47     Y     COMMAND.COM COPY ## 1
 5   105E   M    1064       64  0F47     Y     C:\tb\TSR1.EXE
 6   1063   M    1064      256  0F47     N     C:\tb\TSR1.EXE
 7   1074   M    107B       80  0F47     Y     C:\DOS33\GRAFTABL.COM
 8   107A   M    107B     1344  0F47     N     C:\DOS33\GRAFTABL.COM
 9   10CF   M    10D6       80  10D6     N     COMMAND.COM COPY ## 2
10   10D5   M    10D6     3376  10D6     N     COMMAND.COM COPY ## 2
11   11A9   M    10D6      160  10D6     Y     COMMAND.COM COPY ## 2
12   11B4   M    11BA       64  10D6     Y     C:\tb\MCB3.EXE
13   11B9   Z    11BA   582752  10D6     N     C:\tb\MCB3.EXE
=============================================================================
```

รูปที่ 17 ส่วนของ COMMAND.COM /P ที่ฝังในหน่วยความจำ

```
=============================================================================
MCB   MCB CHAIN CHAIN    BLOCK PARENT  ENV     OWNER
NO.   SEG  ID.   SEG     SIZE   PSP.   BLOCK
=============================================================================
 1   0973   M    0008    23840  022B     N     IBMDOS.COM/MSDOS.SYS
 2   0F46   M    0F47     3376  0F47     N     COMMAND.COM COPY ## 1
 3   101A   M    0000       48  F000     N     FREE MEMORY CONTROL BLOCK
 4   101E   M    0F47     1008  0F47     Y     COMMAND.COM COPY ## 1
 5   105E   M    1064       64  0F47     Y     C:\tb\TSR1.EXE
 6   1063   M    1064      256  0F47     N     C:\tb\TSR1.EXE
 7   1074   M    107B       80  0F47     Y     C:\DOS33\GRAFTABL.COM
 8   107A   M    107B     1344  0F47     N     C:\DOS33\GRAFTABL.COM
 9   10CF   M    10D6       80  10D6     N     COMMAND.COM COPY ## 2
10   10D5   M    10D6     3376  10D6     N     COMMAND.COM COPY ## 2
11   11A9   M    10D6      160  10D6     Y     COMMAND.COM COPY ## 2
12   11B4   M    11BB       80  10D6     Y     C:\micor\KULIST.EXE
13   11BA   M    11BB    70656  10D6     N     C:\micor\KULIST.EXE
14   22FB   M    2302       80  2302     N     COMMAND.COM COPY ## 3
15   2301   M    2302     3376  2302     N     COMMAND.COM COPY ## 3
16   23D5   M    2302      160  2302     Y     COMMAND.COM COPY ## 3
17   23E0   M    23E6       64  2302     Y     C:\MCB3.EXE
18   23E5   Z    23E6   508320  2302     N     C:\MCB3.EXE
=============================================================================
```

รูปที่ 18 ขณะที่อยู่ในโปรแกรม KULIST.EXE แล้วออกสู่ดอสเชลล์